Architektur moderner RISC-Mikroprozessoren

Eidgenössische
Technische Hochschule
Zürich

Departement Informatik
Institut für
Computersysteme

Hans Eberle

Architektur moderner RISC-Mikroprozessoren

June 1997

269

Author's address:

Hans Eberle
Institut für Computersysteme
ETH Zentrum
CH-8092 Zürich
Switzerland

e-mail: eberle@inf.ethz.ch

This report is available via anonymous ftp from ftp.inf.ethz.ch
as pub/publications/tech-reports/2xx/269.ps

Architektur moderner RISC-Mikroprozessoren

Hans Eberle

Institut für Computersysteme
ETH Zürich

Zusammenfassung: Seit der Entwicklung des ersten Mikroprozessors vor rund 25 Jahren konnte die Prozessorleistung kontinuierlich um jährlich 50 % verbessert werden. Diese Leistungsverbesserung ist im wesentlichen den Fortschritten der Halbleiterindustrie zu verdanken, welche den Entwicklern von integrierten Schaltungen einerseits immer schnellere Transistoren und andererseits eine immer grössere Anzahl Transistoren auf einem Chip zur Verfügung stellt. Damit kann die Prozessorleistung erhöht werden, ohne dass Änderungen an der Prozessororganisation nötig sind: die höhere Schaltgeschwindigkeit erlaubt es, Instruktionen schneller auszuführen, und die höhere Integrationsdichte kann dazu benutzt werden, breitere Datenpfade zu realisieren und damit breitere Operandenworte zu verarbeiten. Daneben sind aber auch architektonische Erweiterungen wesentlich mitverantwortlich an der Erhöhung der Rechenleistung. Die beiden wichtigsten Techniken, um die Arbeitsgeschwindigkeit von Mikroprozessoren weiter zu erhöhen, sind die Pipelineverarbeitung und die Verwendung von Cachespeichern. Die Pipelineverarbeitung erlaubt es, mehrere Operationen parallel auszuführen, während Cachespeicher einen schnelleren Zugriff auf Operationen und Operanden ermöglichen.

In diesem Artikel werden die erwähnten Techniken besprochen und deren Anwendung in modernen RISC-Mikroprozessoren aufgezeigt.

Schlüsselwörter: RISC, Mikroprozessoren, Pipelineverarbeitung, Cachespeicher.

Abstract: Since the introduction of the first microprocessor some 25 years ago processor performance has been improved by 50 % per year. This improvement is mainly the result of advances made by the semiconductor industry, which is producing faster and faster transistors and integrating more and more transistors on a single chip. Thus, processor performance can be improved without changing the organization of the processor: higher switching speeds make it possible to execute instructions faster and higher integration density allows to build wider data paths and, with it, to process wider operand words. However, enhanced processor architectures have also contributed significantly to higher processor performance. The two most important techniques to further improve processor performance are pipelining and caching. Pipelining makes it possible to execute several instructions in parallel and caching allows for shorter access times to operations and operands.

This paper discusses the mentioned techniques and their application in modern RISC microprocessors.

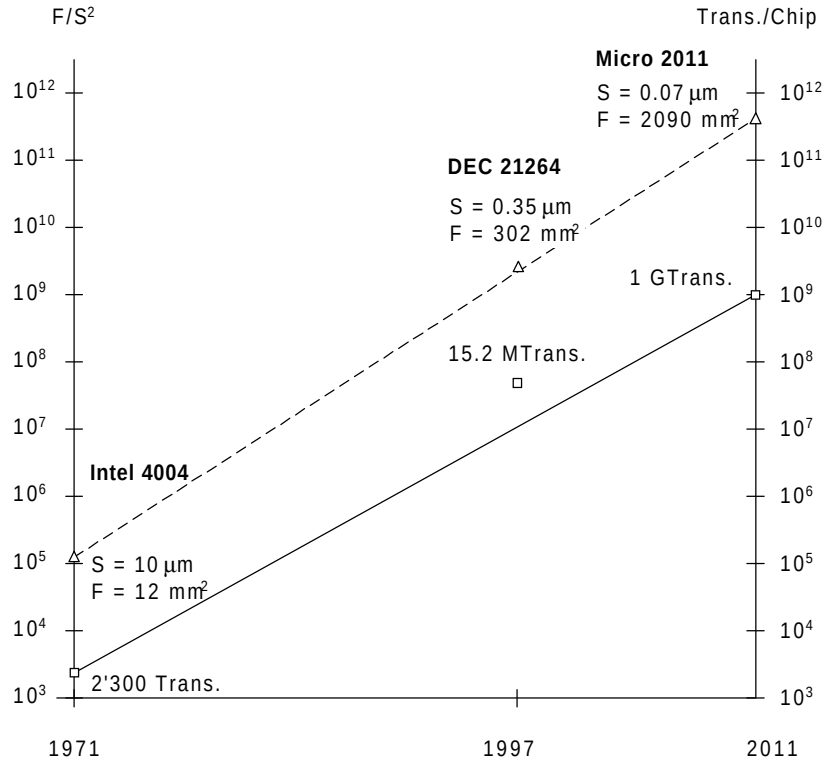
Keywords: RISC, Microprocessors, Pipelining, Cache Memories.

1. Technologische Fortschritte

Die Fortschritte in der Computerindustrie sind vor allem auf technologische Fortschritte der Halbleiterindustrie zurückzuführen. Der mit Abstand wichtigste Technologietrend ist die kontinuierliche Verkleinerung der Strukturgrössen von integrierten Schaltungen. Kleinere Strukturen sind jedoch nur ein Grund, weshalb immer mehr Komponenten auf einem Chip integriert werden können. Weiter gelingt es nämlich den Halbleiterherstellern, Chips mit immer grösserer Grundfläche fehlerfrei herzustellen. Dies ist möglich, indem immer reinere Fabrikationsanlagen für die Verarbeiten der Chipwafer verwendet werden. Die Verkleinerung der Struktu-

ren und Vergrösserung der Chipfläche tragen multiplikativ zu der jährlich fünfzigprozentigen Erhöhung der Anzahl Komponenten bei, welche auf einem Chip integriert werden können.

Figur 1 zeigt den Verlauf dieser Entwicklung anhand konkreter Beispiele. Den Anfang macht der erste, von der Firma Intel im Jahre 1971 auf den Markt gebrachte Mikroprozessor mit der Bezeichnung 4004 [3, 18]. Dieser Chip enthielt 2'300 Transistoren mit einer Strukturgrösse von 10 µm auf einer Chipfläche von 12 mm^2. Der heute leistungsfähigste Mikroprozessor ist der von Digital Equipment entwickelte Alpha-Prozessor 21264 [5], welcher auf einer Chipfläche von 302 mm^2 15.2 Millionen Transistoren mit einer Strukturgrösse von 0.35 µm enthält.

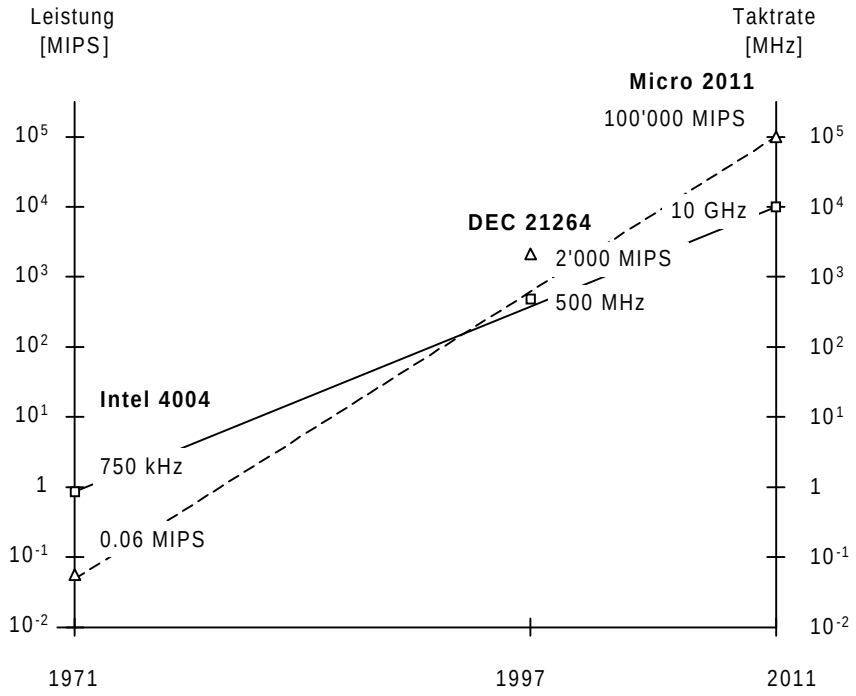


Figur 1: Integrationsdichte (Strukturgrösse S, Chipfläche F)

Wie Figur 2 zeigt, verläuft die Entwicklung der Prozessorleistung in analoger Weise, d.h. auch diese weist eine Verbesserung von jährlich 50 % auf. Während der 4004 einen Durchsatz von 60'000 Instruktionen pro Sekunde erreichte, weist der 21264 einen solchen von 2'000 Millionen Instruktionen pro Sekunde (MIPS) auf. In diesem Zusammenhang interessant ist eine Gegenüberstellung dieser Werte mit den Taktraten. Der 4004 wird mit einer Taktrate von 750 kHz betrieben, wobei zur Ausführung einer Instruktion je nach Instruktionstyp 8 oder 16 Taktzyklen benötigt werden. Instruktionen können nicht überlappt ausgeführt werden, das heisst eine Instruktion wird vollständig abgearbeitet, bevor eine neue Instruktion zur Ausführung gebracht wird. Der 21264 wird mit einer Frequenz von 500 MHz getaktet, wobei in jedem Takt bis zu vier Instruktionen zur Ausführung gebracht werden können - eine 600-MHz-Version des 21264 wurde an der diesjährigen International Solid State Circuits Conference vorgestellt [10].

Es kann erwartet werden, dass sich die aufgezeigten Trends in ähnlicher Weise auch in die nahe Zukunft fortsetzen werden, wobei aber die Grenzen heute eingesetzter Technologien langsam absehbar werden. So skizzierte Intel aus Anlass des 25-jährigen Jubiläums des Mikroprozessors den Micro 2011, einen hypothetischen Prozessor wie er im Jahre 2011, also 40 Jahre nach der Entwicklung des 4004 aussehen dürfte: auf einem 2100 mm^2 grossen Chip werden sich 1 Milliarde Transistoren mit einer Strukturgrösse von 0.07 µm befinden; bei einer Taktfrequenz von 10 GHz wird der Prozessor eine Leistung von 100'000 MIPS erbringen [6].

Die Erhöhung der Taktrate, ermöglicht durch immer kleinere und damit schneller schaltende Transistoren, ist nur ein Faktor, welcher die erzielte Verbesserung der Prozessorleistung erklärt – wie Figur 2 zeigt, hat die Rechenleistung wesentlich mehr zugenommen als die Taktrate. Die insgesamt erzielten Leistungsverbesserungen sind weiter auch das Resultat neuartiger Prozessorarchitekturen, welche erst durch die erhöhte Integrationsdichte möglich wurden. So ist es heute möglich, einerseits Speicher auf dem Prozessorchip selbst zu inte-



Figur 2: Rechenleistung von Mikroprozessoren

grieren und damit die Speicherzugriffszeit zu verkürzen, und andererseits Funktionseinheiten in mehrfacher Ausführung zu realisieren und damit Instruktionen parallel auszuführen. Diese Techniken sollen in den folgenden Abschnitten behandelt werden.

2. Leistungsfaktoren

Ein wesentliches Kriterium zur Beurteilung der Leistungsfähigkeit einer Prozessorarchitektur ist die Zeit, welche zur Ausführung eines Programms benötigt wird[1]. Wir wollen im folgenden die Faktoren, welche die Ausführungszeit bestimmen, etwas ausführlicher betrachten, da sich anhand dieser die Architektur eines Prozessors besser beurteilen lässt. Die Ausführungszeit lässt sich wie folgt berechnen:

$$\textit{Ausführungszeit} = \textit{Anzahl Instruktionen} \times \frac{\textit{Taktzyklen}}{\textit{Instruktion}} \times \frac{\textit{Zeitspanne}}{\textit{Taktzyklus}}$$

$$= \textit{Anzahl Instruktionen} \times \textit{CPI} \times \textit{Taktperiode}$$

$$= \textit{Anzahl Instruktionen} \times \textit{MIPS}^{-1}.$$

Die drei Faktoren, welche also die Ausführungszeit bestimmen, sind die “Zeitspanne pro Taktzyklus”, die Anzahl “Taktzyklen pro Instruktion” und die “Anzahl Instruktionen pro Programm”. Mit “Zeitspanne pro Taktzyklus” ist die Taktperiode, beziehungsweise der Kehrwert der Taktfrequenz gemeint. Der Faktor “Taktzyklen pro Instruktion” gibt an, wie viele Taktzyklen im Durchschnitt für die Ausführung einer Instruktion benötigt werden; oft wird dieser Faktor auch mit CPI (*Cycles per Instruction*) oder TPI (*Ticks per Instruction*) bezeichnet. Mit “Anzahl Instruktionen” ist die Anzahl der tatsächlich ausgeführten Instruktionen gemeint. Der Kehrwert des Produkts von CPI und Taktperiode wird öfters auch mit dem wenig aussagekräftigen, da zu optimistischen Leistungsmass MIPS angegeben.

Will man die Leistung eines Prozessors erhöhen, so muss mindestens einer der drei Faktoren verkleinert werden. Dabei können diese in der Regel nicht unabhängig voneinander verändert werden. So hat beispiels-

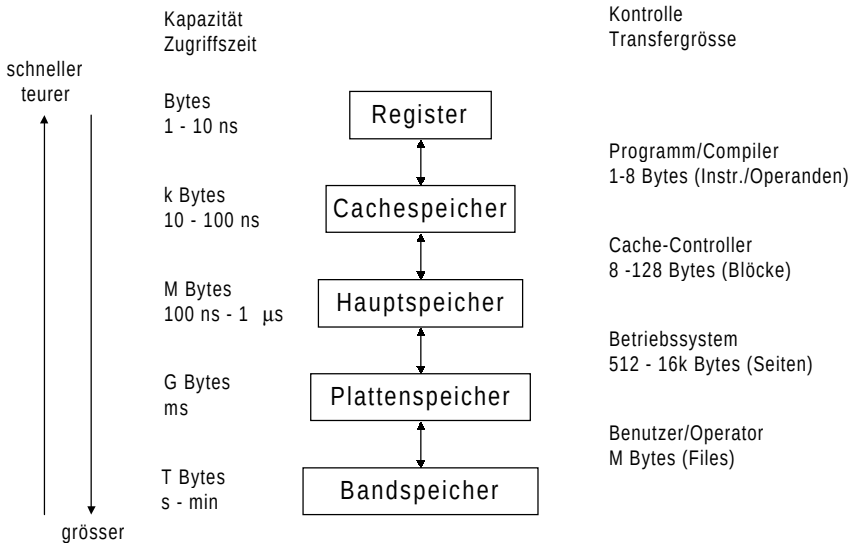
[1] Ein weiteres Kriterium wäre bespielsweise die Grösse des Programmcodes, die heute jedoch dank kostengünstiger Speicherchips weniger ins Gewicht fällt.

weise der Instruktionssatz Einfluss darauf, wie viele Instruktionen zur Ausführung eines Programms benötigt werden, aber auch darauf, wie viele Taktzyklen durchschnittlich für die Ausführung einer Instruktion gebraucht werden. Die Architektur des Prozessors wiederum hat Einfluss auf die Anzahl benötigter Taktzyklen pro Instruktion und auf die Länge der Taktperiode. Die Halbleitertechnologie schliesslich bestimmt die Schaltgeschwindigkeit der einzelnen Komponenten und damit die maximal erreichbare Taktrate.

3. Speicherhierarchie

Die Anbindung des Prozessors an den Speicher wird oft als *von-Neumann-Flaschenhals* bezeichnet, da sämtliche vom Prozessor verarbeiteten Operationen und Operanden durch diesen hindurch müssen. War es früher möglich, den Hauptspeicher direkt an den Prozessor anzuschliessen, sind heute ausgeklügelte Speicherhierarchien nötig, um den Prozessor genügend schnell mit Instruktionen und Daten aus dem Speicher zu versorgen. Dies wurde erforderlich, da die Zugriffsbandbreite und Latenz von Hauptspeicherbausteinen nur wenig verbessert werden konnte und den Anforderungen moderner Prozessoren nicht mehr genügt.

Die wichtigsten Kenngrössen eines Speicherbausteins sind seine Kosten, Geschwindigkeit und Kapazität. Diese drei Grössen lassen sich aber nicht gleichzeitig optimieren: Speicher grosser Kapazität weisen ein relativ günstiges Verhältnis von Kosten pro Speicherbit auf, sind dafür aber langsam; Speicher mit kleiner Kapazität sind verhältnismässig teuer, dafür aber schnell. Diesem Dilemma entgegnet man mit Hilfe einer Hierarchie von Speichern, wie sie in Figur 3 dargestellt ist: die schnellen und teuren Bausteine mit geringer Kapazität befinden sich nahe beim Prozessor, während sich die langsameren und kostengünstigeren Bausteine mit höherer Kapazität weiter entfernt vom Prozessor befinden.



Figur 3: Speicherhierarchie

Der Grund, weshalb Speicher hierarchisch organisiert werden können, liegt in der Lokalität der Speicherzugriffe eines Prozessors begründet. Dabei unterscheidet man zwischen *zeitlicher* und *örtlicher Lokalität*: einerseits ist die Wahrscheinlichkeit gross, dass auf eine Speicherstelle zeitlich mehrfach hintereinander zugegriffen wird, und andererseits ist die Wahrscheinlichkeit gross, dass auf örtlich nahe beieinander liegende Speicherstellen zugegriffen wird. Somit greift also der Prozessor in den meisten Fällen auf den schnellen Speicher und seltener auf den langsameren Speicher zu.

An der Spitze der Speicherhierarchie findet man die Prozessorregister, welche sich auf dem Prozessorchip befinden. Die nächste Hierarchiestufe nimmt der Cachespeicher ein. Bei heutigen Prozessoren besteht diese Stufe wiederum aus einer Hierarchie von Speichern, mit dem Primärcachespeicher auf dem Prozessorchip und dem Sekundär- und eventuell vorhandenen Tertiärcachespeicher ausserhalb des Prozessorchips. Weiter enthält die Hierarchie den Hauptspeicher, auch Primärspeicher genannt, Platten- oder Sekundärspeicher und schliesslich Band- oder Tertiärspeicher. Zu beachten ist der Unterschied in Grösse, Geschwindigkeit und Kosten der verschiedenen Stufen: zwei benachbarte Stufen unterscheiden sich diesbezüglich in der Regel um etwa eine Grössenordnung.

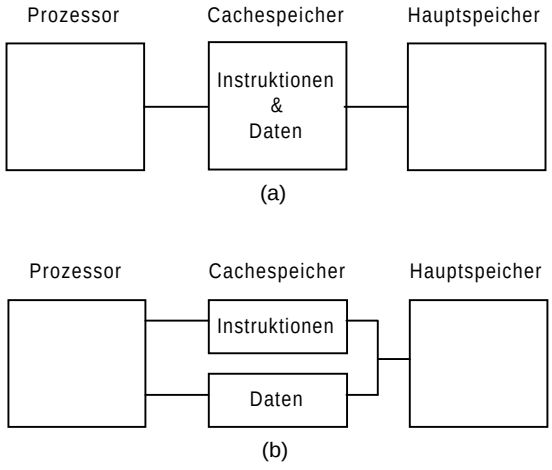
Da sich aus Kapazitätsgründen nicht alle Daten, die bei der Ausführung eines Programms zugegriffen werden, in Prozessornähe befinden können, werden die Daten laufend von einer Hierarchiestufe zur anderen verschoben. Werden Daten zugegriffen, so müssen sie im schlimmsten Fall vom Plattenspeicher in den Hauptspeicher, von dort in den Cachespeicher und schliesslich in die Register geladen werden. Werden Daten auf diese Weise nach oben verschoben, so werden in der Regel als Folge davon Daten nach unten verschoben, um Platz für die neuen Daten zu schaffen.

Je näher sich die Daten beim Prozessor befinden, desto kürzer sind die Zugriffszeiten und desto weniger Zeit steht für die Abwicklung des Zugriffs zur Verfügung, also beispielsweise um zu bestimmen, auf welche Adresse zugegriffen werden soll. Aus diesem Grunde wird die Kontrolllogik für den Cachespeicher mittels schneller Hardware realisiert, da der Prozessor auf diesen in einem Taktzyklus oder ein paar wenigen Taktzyklen zugreifen möchte, wohingegen Zugriffe auf den Plattenspeicher langsamer, unter Softwarekontrolle erfolgen können, da nur schon das Positionieren der Plattenköpfe in der Regel im Bereich einiger Millisekunden liegt und somit Tausende von Prozessortaktzyklen beansprucht.

4. Cachespeicher

Die heute mögliche Integrationsdichte integrierter Schaltungen lässt sich nur mit regulären Strukturen ausnutzen, wie sie für Speicherbausteine benutzt werden. Aus diesem Grunde wird der grösste Anteil von Transistoren auf einem Prozessorchip für die Realisierung von Speicher, d.h. Register und Cachespeicher eingesetzt. So werden beispielsweise von den 15.2 Millionen Transistoren des 21264 “lediglich” deren 6 Millionen für die CPU-Logik benötigt [5].

Grundsätzlich lassen sich Cachespeicher in die beiden, in Figur 4 gezeigten Organisationsformen unterteilen. Befinden sich Instruktionen und Daten in einem gemeinsamen Cache, so spricht man von einer *Princeton-Architektur*, werden die Instruktionen und Daten hingegen in separaten Caches gespeichert, so spricht man von einer *Harvard-Architektur*. Beide Formen haben ihre Vor- und Nachteile. Der wesentliche Vorteil von separaten Caches ist die Möglichkeit, auf Daten und Instruktionen gleichzeitig zugreifen zu können. Der Vorteil eines gemeinsamen Caches ist die flexible Zuteilung von Cachespeicherplatz: benötigt ein Programm mehr Speicherplatz für die Instruktionen als für die Daten (oder umgekehrt), so wird die Speicherzuteilung automatisch entsprechend geändert.

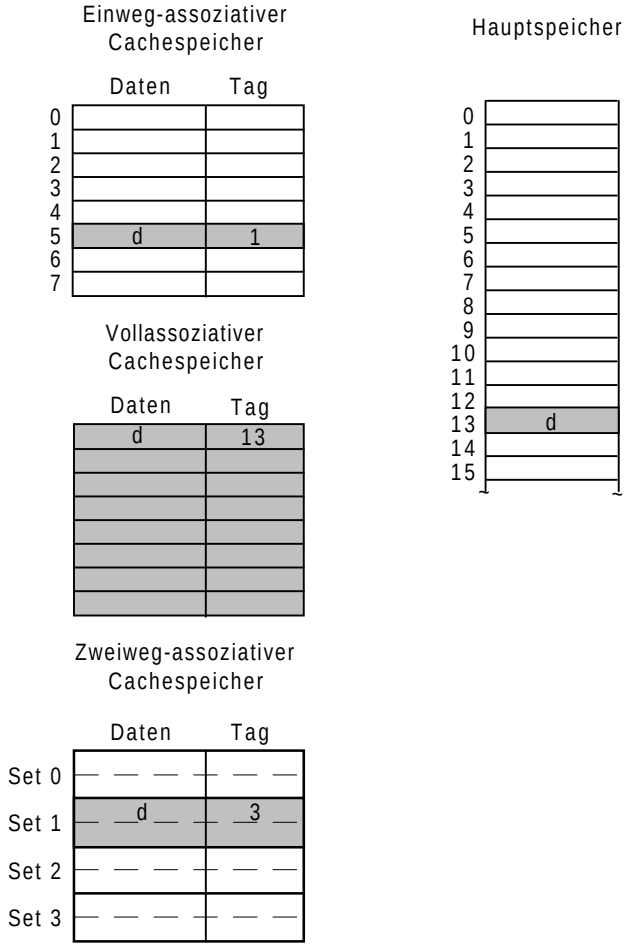


Figur 4: Cachespeicher mit einer Princeton (a) und einer Harvard-Architektur (b)

Ein wesentliches Merkmal eines Cachespeichers ist der *Assoziativitätsfaktor*, der angibt, wie viele Einträge im Cache für die Speicherung eines Blocks aus dem Hauptspeicher in Frage kommen. Bei der einfachsten Organisationsform, dem *einweg-assoziativen* oder *direct-mapped* Cache gibt es für jeden Speicherblock nur gerade einen möglichen Aufenthaltsort. Dabei geschieht die Adressumsetzung wie folgt:

Cacheadresse = Speicheradresse MOD Anzahl Cacheeinträge.

Figur 5 zeigt ein Beispiel: der Speicherblock mit der Adresse 13 wird im einweg-assoziativen Cache an der Adresse 5 (= 13 MOD 8) gespeichert.



Figur 5: Cachespeicher-Organisation

Diese einfache Adressumsetzung hat den Nachteil, dass beim Zugriff auf Hauptspeicheradressen, welche auf die gleiche Adresse im Cache abgebildet werden, der Cacheeintrag laufend überschrieben wird. Würden beispielsweise in Figur 5 abwechslungsweise die Speicherblöcke mit den Adressen 13 und 21 zugegriffen, so würde der Cacheeintrag mit der Adresse 5 immer wieder überschrieben und der Geschwindigkeitsvorteil von Cachezugriffen käme nicht zum tragen.

Solche Konflikte treten mit einem vollassoziativen Cache nicht auf. Wie Figur 5 zeigt, kommt bei dieser Organisationsform jeder Cacheeintrag für die Speicherung eines Speicherblocks in Frage. Wie später gezeigt wird, steigt jedoch mit der Grösse des Assoziativitätsfaktors der Realisierungsaufwand, weshalb in der Praxis günstigere Formen, wie der zweiweg-assoziative Cache gewählt werden. Allgemein gilt, dass es in einem *n*-weg-assoziativen Cache *n* mögliche Einträge für die Speicherung eines Speicherblocks gibt, wobei man sagt, dass die *n* möglichen Einträge zu einem *Set* gehören. Der Index für das in Frage kommende Set wird wie folgt berechnet:

Cacheset = Speicheradresse MOD Anzahl Cachesets.

In Figur 5 ist ein Beispiel für einen zweiweg-assoziativen Cache gezeigt: für den Speicherblock mit der Adresse 13 kommen die beiden Einträge des Sets 1 (= 13 MOD 4) in Frage.

Da die Abbildung von Hauptspeicheradressen auf Cachespeicheradressen nicht eineindeutig ist, d.h. es wie erwähnt für einen Eintrag im Cache mehrere Kandidaten aus dem Hauptspeicher gibt, muss zusätzliche Information abgespeichert werden, um bei einem späteren Lesezugriff die Hauptspeicheradresse eines Eintrags ein-

deutig bestimmen zu können. Die dazu nötige Information wird *Tag* genannt, das aus den höherwertigen Hauptspeicheradressbits besteht. Das Tag für einen einweg-assoziativen Cache wird wie folgt berechnet:

$$Tag = Speicheradresse\ DIV\ Anzahl\ Cacheeinträge.$$

Die Berechnung des Tags für einen mehrweg-assoziativen Cache sieht entsprechend aus:

$$Tag = Speicheradresse\ DIV\ Anzahl\ Cachesets.$$

Es genügt, die höherwertigen Speicheradressbits im Tag zu speichern, da ja die niederwertigen Adressbits der Adresse des Cacheeintrags entsprechen.

Soll geprüft werden, ob sich ein Speicherblock im Cache befindet, so muss das Tag mit der Speicheradresse verglichen werden. Bei einem einweg-assoziativen Cache ist ein einziger Vergleich nötig, wohingegen bei einem *n*-weg-assoziativen Cache *n* Vergleiche nötig sind. Damit sind auch die höheren Kosten eines vollassoziativen Caches erklärt. Darüber hinaus muss eine längere Zugriffszeit in Kauf genommen werden, da nicht nur festgestellt werden muss, ob der Cache den Speicherblock enthält, sondern auch, welcher Eintrag eines Sets den Block enthält.

Das wohl wichtigste Kriterium zur Beurteilung der Leistungsfähigkeit der Architektur eines Cachespeichers ist die *Missrate*, welche angibt, mit welcher Wahrscheinlichkeit ein Speicherzugriff vom Cachespeicher nicht beantwortet werden kann und einen Zugriff auf den Hauptspeicher erfordert. Damit sieht das in Kapitel 2 eingeführte Leistungsmass wie folgt aus:

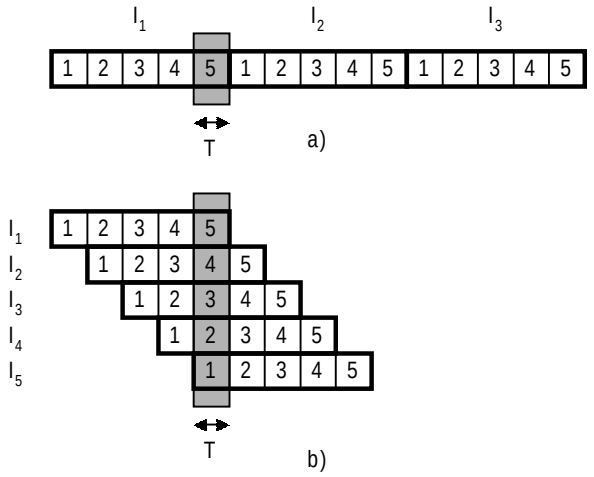
$$Ausführungszeit = Anzahl\ Instruktionen\ x\ (CPI + \frac{Speicherzugriffe}{Instruktion}\ x\ Missrate\ x\ Missverzögerung)\ x\ Taktperiode.$$

Falls ein Zugriff auf den Hauptspeicher erforderlich ist, muss die zusätzliche Verzögerung berücksichtigt werden. Je nach Grösse der *Missrate* und der *Missverzögerung* kann diese die Ausführungszeit wesentlich mitbestimmen. Wichtig ist der Hinweis, dass die Missrate nicht nur von der Organisation des Cachespeichers abhängt, sondern auch von dem ausgeführten Programm selbst.

5. Pipelineverarbeitung

Die *Pipelineverarbeitung* ermöglicht es, mehrere Instruktionen überlappt auszuführen und damit die Arbeitsgeschwindigkeit eines Prozessors zu erhöhen. Ähnlich der Fliessbandverarbeitung in einer Fabrik wird bei der Pipelineverarbeitung die Ausführung einer Instruktion in mehrere Teilschritte zerlegt, womit es möglich wird, gleichzeitig unterschiedliche Teilschritte mehrerer Instruktionen zu bearbeiten.

Figur 6 veranschaulicht die Verarbeitung von Instruktionen mit und ohne Pipelineverarbeitung. In dem angegebenen Beispiel wird die Verarbeitung einer Instruktion in fünf Teilschritte zerlegt. Die in einem Takt geleistete Arbeit ist für beide Organisationsformen hervorgehoben. Die Pipeline enthält also bis zu fünf Instruk-



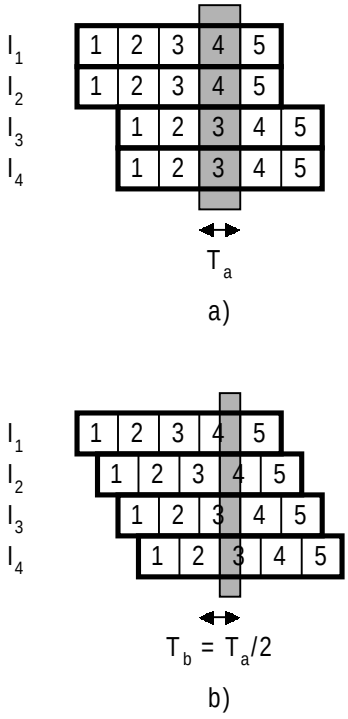
Figur 6: Instruktionsverarbeitung ohne (a) und mit (b) Pipelineverarbeitung

tionen, welche gleichzeitig in Bearbeitung sind. Zu beachten ist, dass die Latenz, d.h. die Zeit, welche benötigt wird, um eine einzelne Instruktion zu verarbeiten, nicht kürzer wird dank Pipelineverarbeitung. Im Gegenteil, Pipelineverarbeitung erfordert normalerweise einen zusätzlichen, wenn auch geringen zeitlichen Mehraufwand. Die Verarbeitungsrate kann hingegen mittels Pipelineverarbeitung drastisch gesteigert werden. In Figur 6a wird alle 5 Zyklen die Ausführung einer Instruktion beendet, wohingegen in Figur 6b pro Zyklus die Ausführung einer Instruktion beendet wird.

Man nennt die Zeitdauer, welche eine Instruktion in einer Pipelinestufe verbringt, *Maschinenzyklus*. Ein Maschinenzyklus entspricht in der Regel einem *Taktzyklus*, kann aber auch, wie später gezeigt wird, durch mehrere Taktzyklen realisiert werden. Da die Pipelinestufen synchron getaktet werden, gibt die langsamste Stufe den Takt vor. Beim Entwurf einer Pipeline wird man also bemüht sein, Pipelinestufen mit möglichst ausgeglichener Länge zu realisieren. Dank Pipelineverarbeitung kann der Durchsatz unter idealen Bedingungen um die Anzahl Pipelinestufen erhöht werden. Dieses Ideal wird in der Praxis jedoch aus mehreren Gründen nicht erreicht. So sind die Stufen einer Pipeline typischerweise nicht perfekt ausgeglichen. Weiter ist es nicht immer möglich, gleichzeitig in allen Pipelinestufen nützliche Arbeit zu verrichten, sei es weil Ressourcenkonflikte auftreten oder Instruktionen voneinander anhängig sind. Und schliesslich wird zusätzliche Zeit benötigt, um die Daten beim Übergang von einer Pipelinestufe zur nächsten zwischenzuspeichern.

Neuere Prozessoren weisen sogar CPI-Werte von weniger als 1 auf, was bedeutet, dass der Durchsatz mehrere Instruktionen pro Takt beträgt. Architekturen mit solchen Leistungsmerkmalen verwenden entweder eine *Superpipeline* oder eine *superskalare Pipeline*. Ein Prozessor mit einer superskalaren Pipeline erlaubt es, mehr als eine Instruktion pro Takt zur Ausführung zu bringen. Damit dies möglich ist, müssen die zur Ausführung der Instruktionen nötigen Ressourcen, wie Funktionseinheiten und Datenpfade in mehrfacher Ausführung vorhanden sein. Vergleichbare Leistungsverbesserungen lassen sich auch mit einer Superpipeline erreichen. Dabei wird eine Pipelinestufe in weitere Stufen aufgeteilt mit dem Ziel, die Pipeline schneller zu takten und damit den Durchsatz zu erhöhen. Die Klassifizierung einer Pipeline als Superpipeline ist in der Praxis nicht immer einfach, da diese gerade so gut als tiefe Pipeline mit fein-granularen Stufen betrachtet werden kann. Figur 7 veranschaulicht diese beiden Techniken. Beide gezeigten Pipelineorganisationen weisen einen CPI-Wert von 0.5 auf.

Es ist wichtig festzuhalten, dass die parallele Verarbeitung von Instruktionen mittels Pipelineverarbeitung für den Programmierer unsichtbar bleibt. Die Instruktionen werden zwar überlappt ausgeführt, jedoch in ihrer ursprünglichen sequentiellen Ordnung abgearbeitet.



Figur 7: Superskalare Pipeline (a) Superpipeline (b)

6. RISC-Pipeline

Konventionelle Prozessoren mit einer *CISC[2]-Architektur* benutzen tiefe, das heisst eine grosse Anzahl Stufen enthaltende Pipelines. Diese lassen sich verhältnismässig schlecht ausnutzen und weisen eine Durchsatzrate von typischerweise etwa 10 CPI auf. Moderne Prozessoren mit *einer RISC[3]-Architektur* hingegen verwenden Pipelines mit weniger Stufen und weisen Durchsatzraten von typischerweise weniger als 1 CPI auf. Mittels folgender Techniken wird diese Erhöhung des Durchsatzes erreicht:

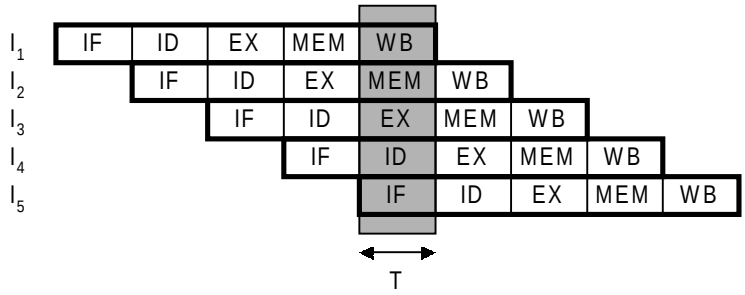
- *1 CPI*: Das ursprüngliche und wohl wichtigste Ziel bei der Entwicklung von RISC-Architekturen war eine Verarbeitungsrate von einer Instruktion pro Maschinenzyklus, d.h. 1 CPI. Erreicht wird dies mittels eines - wie mit der Bezeichnung RISC angedeutet wird - reduzierten, einfachen Instruktionssatzes.
- *Fixe Instruktionslänge*: Ein Instruktionsformat mit einer fixen Länge von typischerweise 32 Bits erleichtert das Dekodieren der Instruktionen, womit es leichter möglich ist, eine Instruktion pro Taktzyklus zur Ausführung zu bringen.
- *Wenige, einfache Operationen*: Im Gegensatz zu den komplexen Instruktionen eines CISC-Prozessors, kennt ein RISC-Prozessor nur einfache (und ursprünglich wenige) Instruktionen, was die Pipeline vereinfacht und höhere Taktraten zulässt. Da der Arbeitsaufwand für alle Instruktionen eines RISC-Prozessors etwa gleich gross ist, können Instruktionen einfacher und effizienter überlappt verarbeitet werden.
- *Load-/Store-Architektur*: Der Instruktionssatz eines RISC-Prozessors zerfällt in drei Klassen von Instruktionen: arithmetische/logische Operationen, Speicherzugriffe und Sprungbefehle. Operationen können nur Operanden verwenden, welche sich in Registern befinden. Der Zugriff auf den Speicher ist den beiden Instruktionen *Load* und *Store* vorbehalten. Durch diese Entkopplung von Speicherzugriffen und eigentlichen Operationen ist es wiederum möglich, die Pipelineverarbeitung zu optimieren.
- *Grosser Registersatz*: Um die Anzahl von Load- und Store-Instruktionen zu reduzieren, welche benötigt werden, um Operanden im Speicher zuzugreifen, wird eine möglichst grosse Anzahl von Registern bereitgestellt. Typischerweise enthalten RISC-Prozessoren je 32 Register für Integer- und Gleitkomma-Operanden.
- *Einfache Adressiermodi*: Komplizierte Adressiermodi erfordern mehr Aufwand bei der Berechnung der Adresse, was eine längere Taktperiode zur Folge hat. Im Extremfall besitzt ein RISC-Prozessor nur den einen Modus *Register-relativ*, bei dem die Speicheradresse gleich der Summe einer sich in einem Register befindenden Basisadresse und einem in der Instruktion enthaltenen Offset ist.
- *Delayed Branches*: Da bei einem bedingten Sprungbefehl erst die Bedingung ausgewertet werden muss, bevor feststeht, welche Instruktion als nächste ausgeführt wird, entstehen ohne zusätzliche Vorkehrungen Verzögerungen in einer Pipeline. Das Konzept des Delayed Branches lässt es zu, dass der Prozessor ohne zu warten mit der Ausführung der Instruktionen weiterfährt, die dem Sprungbefehl unmittelbar folgen. Dies bedingt, dass der Compiler oder Programmierer solche Instruktionen im sogenannten *Delay Slot* nach dem Sprungbefehl unterbringt, welche ausgeführt werden dürfen unabhängig vom Ausgang des Sprungbefehls.
- *Optimierende Compiler*: Um die Ressourcen einer RISC-Pipeline optimal ausnützen zu können, werden optimierende Compiler benötigt, wie anhand der Delayed Branches gezeigt wurde.

In etwas vereinfachter Form lässt sich eine typische RISC-Pipeline wie in Figur 8 beschreiben. Die Pipeline besteht aus den folgenden fünf Stufen:

- *IF* (Instruction Fetch): die Instruktion wird aus dem Speicher geholt.
- *ID* (Instruction Decode): die Instruktion wird dekodiert und die Quelloperanden werden aus den Prozessorregistern gelesen.

[2] Complex Instruction Set Computer
[3] Reduced Instruction Set Computer

- *EX* (Execute): die Instruktion wird ausgeführt.
- *MEM* (Memory Reference): falls es sich um eine Load- oder Store-Instruktion handelt, wird der Speicher zugegriffen.
- *WB* (Write Back): der berechnete Zieloperand wird in ein Register geschrieben.

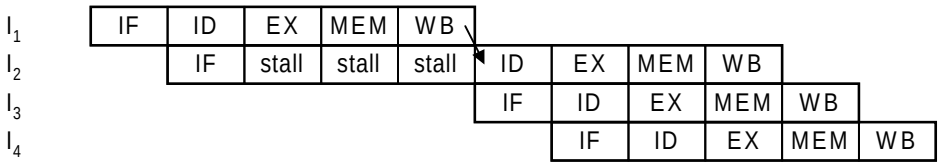


Figur 8: RISC-Pipeline

Trotz der angegebenen Techniken ist es nicht immer möglich, in allen Stufen einer RISC-Pipeline nützliche Arbeit zu verrichten. Die Folge sind Verzögerungen, auch *Pipeline Stalls* genannt. Solche treten zum Beispiel auf, wenn zwischen den Operanden aufeinanderfolgender Instruktionen Abhängigkeiten bestehen. Das folgende Beispiel illustriert einen solchen Fall:

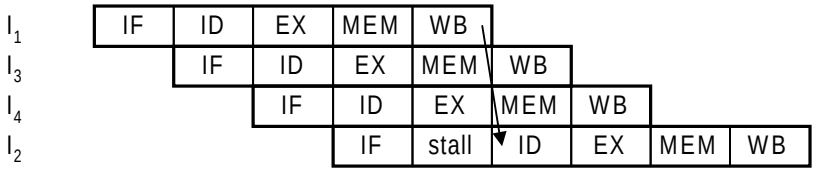
I_1:	Add R0, R1, R2	*R0 := R1 + R2*
I_2:	Sub R3, R0, R4	*R3 := R0 - R4*
I_3:	Mul R5, R6, R7	*R5 := R6 * R7*
I_4:	Mul R8, R9, R10	*R8 := R9 * R10*

Zwischen den Instruktionen I_1 und I_2 besteht eine Abhängigkeit, da das Zielregister R0 von I_1 ebenfalls Quellregister von I_2 ist. Die entstehenden Verzögerungen sind in Figur 9 gezeigt: I_2 darf das Register R0 in der Pipelinestufe ID erst lesen, nachdem dieses von I_1 in der Stufe WB geschrieben worden ist. Damit wird die Ausführung von I_2 und aller nachfolgenden Instruktionen um drei Taktzyklen verzögert.



Figur 9: Verzögerungen in Pipelines

Bei genauerer Betrachtung der angegebenen Instruktionssequenz sieht man, dass die von I_3 und I_4 verwendeten Quellregister weder das Zielregister von I_1 noch von I_2 sind. Würde man also die Sequenz umordnen und die Instruktionen in der in Figur 10 angegeben Reihenfolge abarbeiten, so könnten zwei Wartezyklen verhindert werden.



Figur 10: Umordnen von Instruktionen

Für einfache Pipelineorganisationen kann das Umordnen von Instruktionen in der gezeigten Weise statisch durch einen optimierenden Compiler erfolgen. Die komplexen superskalaren Pipelines der heutigen RISC-Prozessoren erfordern jedoch aufwendigere Techniken, welche in Hardware, als Teil des Steuerwerks eines Prozessors realisiert werden müssen. Dabei werden die Instruktionen, respektive die zu deren Ausführung erforderlichen Teilschritte dynamisch, zur Laufzeit umgeordnet, um möglichst viele Funktionseinheiten gleich-

zeitig zu beschäftigen. Um die Komplexität dieser Aufgabe zu verdeutlichen, sei der 21264 erwähnt, der insgesamt bis zu 80 Instruktionen gleichzeitig in Bearbeitung haben kann [5].

Steuerwerke, welche die nicht-sequentielle Ausführung von Instruktionen ermöglichen, können auf zwei Arten konzipiert werden: entweder wird, wie man sagt, der *Kontrollfluss* oder der *Datenfluss* gesteuert. Die Kontrollflusssteuerung ist zentral realisiert und bestimmt während der Dekodierphase, welche Instruktionen zur Ausführung gebracht werden können. Aufgrund der Abhängigkeiten und der vorhandenen Ressourcen wird zu diesem Zeitpunkt bestimmt, ob eine Instruktion zur Ausführung gebracht werden kann oder zurückgehalten werden muss. Im Gegensatz zur Kontrollflusssteuerung gibt es bei der Datenflusssteuerung keine zentrale Instanz, welche den Ablauf der Instruktionsausführung kontrolliert; vielmehr ist die Steuerung dezentral auf die Funktionseinheiten verteilt. Die Instruktionen verlassen die Dekodierphase, sobald sie dekodiert worden sind, und werden in Pufferspeichern, welche sich bei den Funktionseinheiten befinden, zwischengespeichert bis die Operanden verfügbar sind und die Funktionseinheit frei ist.

Viele in modernen Prozessoren verwendeten Techniken wurden bereits vor geraumer Zeit in Mainframe-Rechnern eingesetzt. Dies gilt auch für die nicht-sequentielle Ausführung von Instruktionen. So wurde in der CDC 6600 [16] ein sogenanntes *Scoreboard* für die Realisierung einer Kontrollflusssteuerung verwendet. Ein frühes Beispiel einer Maschine, welche eine Datenflusssteuerung verwendete, war die IBM 360 Modell 91 [17]. Die Steuerung verwendete den nach seinem Entwickler benannten *Tomasulo Algorithmus* und die Pufferspeicher wurden *Reservationsstationen* genannt. Die Begriffe Scoreboard und Reservationsstation finden sich auch heute wieder in den Datenblättern "moderner" Prozessoren.

In einer Diskussion über Pipelines dürfen Verzögerungen infolge von Sprunginstruktionen nicht unerwähnt bleiben. Ein Sprung führt dazu, dass Instruktionen nicht mehr weiter von sequentiellen Adressen geholt werden können. Die IF-Einheit kann in diesem Fall nicht einfach blindlings den Programmzähler inkrementieren, der ja jeweils auf die als nächste zu holende Instruktion zeigt, sondern es muss zuerst der Sprungbefehl interpretiert werden, um den neuen Wert des Programmzählers zu bestimmen. Da also oftmals bis zum Ausführen der Sprunginstruktion nur Spekulationen gemacht werden können, welche Instruktionen als nächste ausgeführt werden, können Verzögerungen bei der überlappten Befehlsausführung in einer Pipeline entstehen. Diverse Techniken wurden entwickelt, um solche Verzögerungen zu reduzieren oder zu verhindern. Für eine ausführliche Behandlung solcher Massnahmen muss auf die Literatur verwiesen werden [4, 8, 15].

7. PowerPC, Alpha und MIPS im Vergleich

In diesem Abschnitt soll die Anwendung der besprochenen Techniken in modernen RISC-Mikroprozessoren vorgestellt werden. Als deren Vertreter wurden die 64-bit Prozessoren PowerPC 620 [9, 12, 20], Alpha 21264 [2, 5, 11, 14] und MIPS R10000 [22] gewählt.

In Tabelle 1 werden die Kenngrössen der Implementationen dieser Prozessoren verglichen [21]. Dabei ist zu berücksichtigen, dass die Markteinführung der angegeben Implementationen zu unterschiedlichen Zeitpunkten erfolgte. Mit 15.2 Millionen Transistoren auf einem rund 3 cm^2 grossen Chip weist der 21264 nicht nur mit Abstand die grösste Anzahl Transistoren, sondern auch die grösste Dichte auf. Um eine hohe Dichte zu ermöglichen, werden nicht nur Transistoren mit kleinen Strukturgrössen benötigt, sondern es müssen auch genügend Verbindungsmöglichkeiten in der Form von Metalllayern vorhanden sein. Dies zeigt beispielsweise der Vergleich des 21264 mit dem R10000: bei gleicher Strukturgrösse und ungefähr gleich grosser Chipfläche finden auf dem 21264 mehr als doppelt so viele Transistoren Platz wie auf dem R10000, da beim 21264 sechs Metalllayer zur Verfügung stehen und beim R10000 nur deren vier.

	PowerPC 620	Alpha 21264	MIPS R10000
Markteinführung	1995	1997	1996
Technologie	0.5 µm CMOS, 4M	0.35 µm CMOS, 6M	0.35 µm CMOS, 4M
Chipgrösse	3.11 cm^2	3.02 cm^2	2.98 cm^2
Anzahl Transistoren	6.9 x 10^6	15.2 x 10^6	6.8 x 10^6
Taktfrequenz	133 MHz	600 MHz	200 MHz
Stromverbrauch	30 W	72 W	~30 W

Tabelle 1: Implementierungscharakteristika

Auffallend sind die unterschiedlichen Werte für Taktfrequenz und maximalen Stromverbrauch. CMOS-Schaltungen haben die Eigenschaft, dass ihr Stromverbrauch praktisch linear mit der Taktfrequenz ansteigt:

der 21264 wird drei mal schneller getaktet als der 620 und R10000 und verbraucht als Folge davon rund zweieinhalb mal soviel Leistung.

Die Leistungsfähigkeit von Prozessoren wird heute üblicherweise mit Hilfe von Benchmark-Programmen des SPEC-Industriekonsortiums *(System Performance Evaluation Cooperative)* gemessen. Wurden früher vorwiegend synthetische Programme verwendet, so enthalten die SPEC-Benchmarks regulär verwendete Programme wie einen Compiler, ein Programm zur Minimierung Bool'scher Ausdrücke, ein Tabellenkalkulationsprogramm und andere Programme, die die arithmetische Leistungsfähigkeit messen. Mit diesen Messungen erhält der Endbenutzer ziemlich aussagekräftige Beurteilungsgrössen. Tabelle 2 enthält die für die verglichenen Prozessoren erhaltenen Grössen, wobei diese in zwei Gruppen unterteilt sind: *int95* enthält Programme, die vor allem mit ganzen Zahlen operieren, und *fp95* enthält Programme, welche vor allem mit Gleitkommazahlen operieren.

	PowerPC 620	Alpha 21264	MIPS R10000
SPEC int95	6	~40	10.7
SPEC fp95	6	~60	13.8

Tabelle 2: Leistungszahlen

Alle drei Prozessoren besitzen Primärcachespeicher auf dem Prozessorchip, doch weisen diese unterschiedliche Kapazitäten und Organisationsformen auf, wie dies Tabelle 3 belegt. Insbesondere fällt auf, dass der PowerPC 620 über Daten- und Instruktionscaches mit grossen Assoziativitätsfaktoren verfügt. Wie in Abschnitt 3 erwähnt wurde, reduziert ein hoher Assoziativitätsfaktor in der Regel die Missrate, verlängert aber die Zugriffszeiten. Da der 620 mit einer wesentlich niedrigeren Taktrate als der 21264 und R10000 betrieben wird, dürfte mehr Zeit zur Verfügung stehen, um auf den Cache zuzugreifen.

	PowerPC 620	Alpha 21264	MIPS R10000
Primärcache			
Harvard/Princeton	Harvard	Harvard	Harvard
Assoziativitätsfaktor (D/I)	8/8	2/2	2/2
Kapazität (D/I)	32 kByte/32 kByte	64 kByte/64 kByte	32 kByte/32 kByte
Sekundärcache			
Harvard/Princeton	Princeton	Princeton	Princeton
Assoziativitätsfaktor	1	1	2
Kapazität	bis 128 MByte	bis 16 MByte	bis 16 MByte

Tabelle 3: Cacheorganisation

Sämtliche aufgeführten Prozessoren besitzen die nötige Interfacelogik, um auf einen externen Sekundärspeicher zugreifen zu können. Während die internen Primärcaches eine Harvard-Organisation aufweisen, sind die externen Sekundärspeicher mit einer Princeton-Organisation aufgebaut, was deren Realisierung kostengünstiger gestaltet, da nur eine Speicherbank vorgesehen werden muss, was bei 128-bit-breiten Datenpfaden ins Gewicht fällt.

Tabelle 4 vergleicht die Pipelineorganisation der betrachteten Prozessoren. Alle drei verwenden eine superskalare Pipeline[4]. Der 620 und 21264 verfügen über insgesamt sechs Funktionseinheiten, wobei gleichzeitig mit der Ausführung von bis zu vier Instruktionen begonnen werden kann. Beim R10000 sind fünf Funktions-

	PowerPC 620	Alpha 21264	MIPS R10000
Funktionseinheiten	6	6	5
Parallelität	4	4	5
Pipelinestufen			
Gleitkomma-Instruktionen	6	10	7
Load-/Store-Instruktionen	4/5[1]	7	6
Integer-Instruktionen	4	7	5

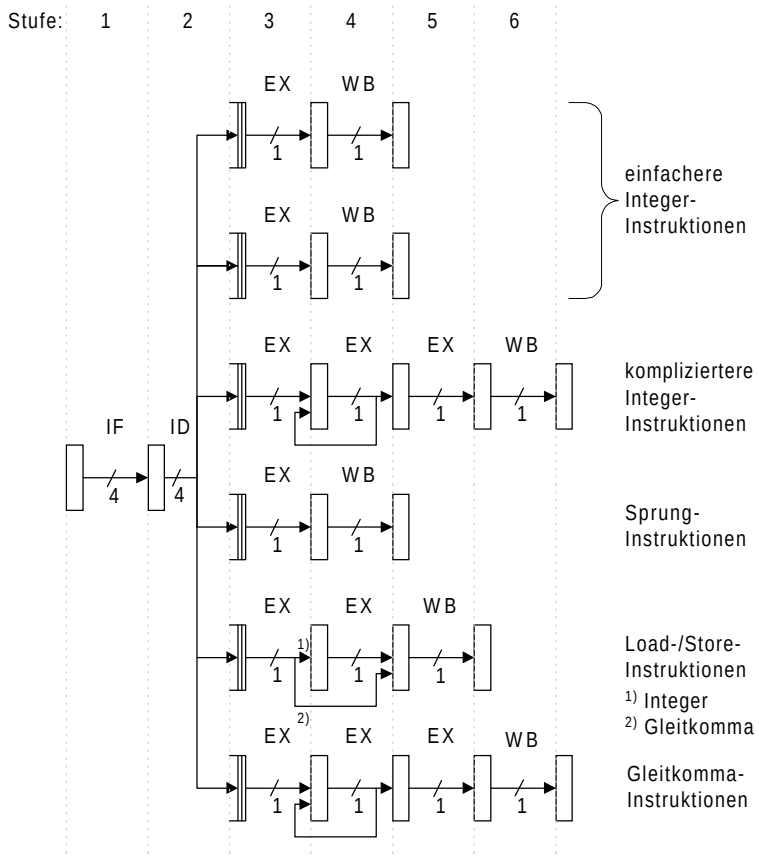
Tabelle 4: Pipelineorganisation

[4] Ein Beispiel einer Superpipeline findet man im R4400, einem Vorgänger des R10000.

einheiten vorhanden, die alle gleichzeitig mit der Ausführung einer Instruktion beginnen können. Bei allen drei Organisationsformen sind Reservationsstationen vorhanden, so dass Instruktionen nicht in der vom Programm vorgegebenen Reihenfolge ausgeführt werden müssen.

Die Anzahl Pipelinestufen ist gemäss den drei Instruktionstypen aufgegliedert. Die angegebenen Werte gelten für die meisten Instruktionen der jeweiligen Kategorie, jedoch nicht für gewisse aufwendigere Instruktionen, wie Multiplikation oder Division. Es fällt auf, dass die Pipeline des 21264 die längste ist, was mit der hohen Taktrate erklärt werden kann, die der einzelnen Stufe weniger Verarbeitungszeit gibt und damit eine höhere Anzahl Stufen verlangt.

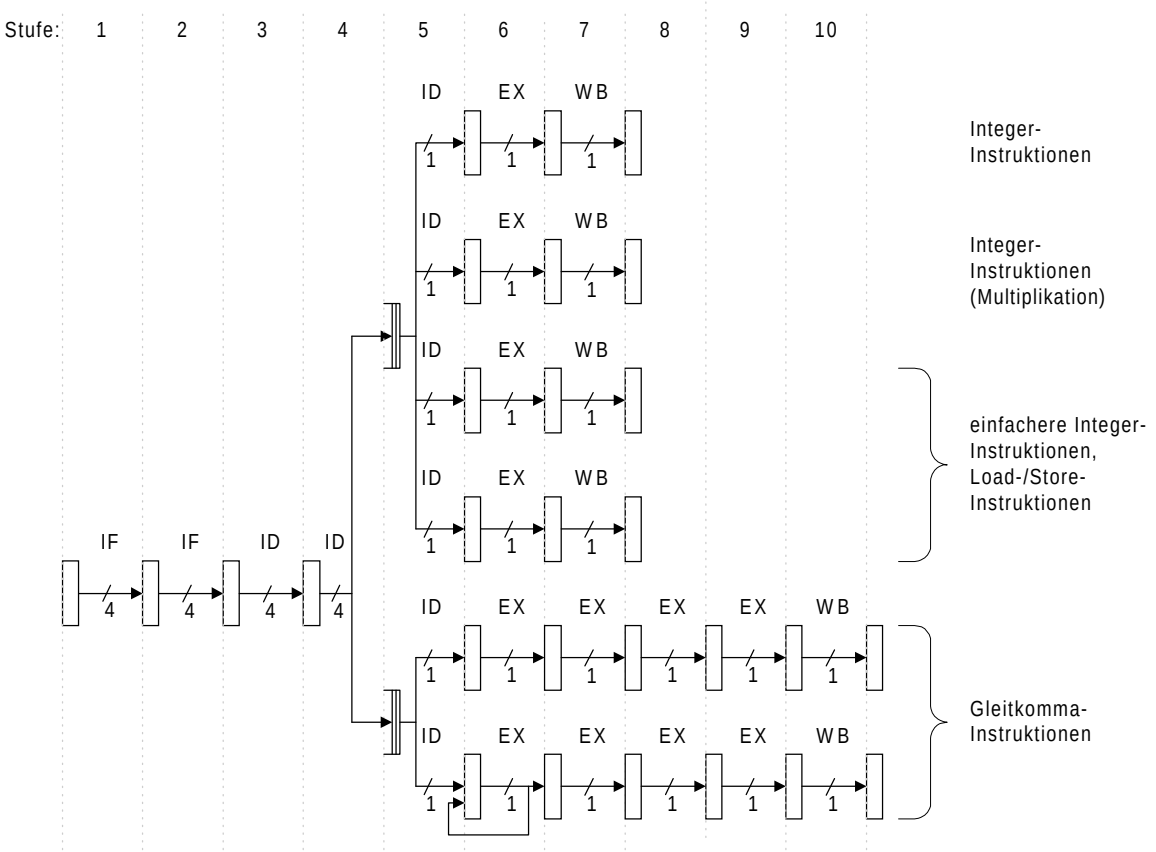
In den folgenden Figuren sind die Pipelines der drei untersuchten Prozessoren im Detail angegeben. Die Aufgaben der einzelnen Pipelinestufen sind mit den aus Abschnitt 5 bekannten Bezeichnungen versehen. Den Anfang macht der 620, dessen Datenpfad in Figur 11 gezeigt ist und wie folgt unterteilt ist. In Stufe 1 werden jeweils vier Instruktionen pro Takt aus dem I-Cache geholt. In Stufe 2 werden die Instruktionen dekodiert. Weiter wird in dieser Stufe entschieden, welche Instruktionen im nächsten Takt den Funktionseinheiten übergeben werden können und welche Instruktionen in Reservationsstationen zwischengespeichert werden müssen, da mit deren Ausführung zugewartet werden muss, beispielsweise da die Operanden noch nicht verfügbar sind. Dabei verfügt jede Funktionseinheit über ihre eigene Reservationsstation. Insgesamt sind sechs Funktionseinheiten vorhanden: zwei identische Integer-Einheiten für einfachere Operationen wie Addition und Subtraktion, eine Integer-Einheit für kompliziertere Berechnungen wie Multiplikation und Division, eine Gleikomma-Einheit, eine Einheit für das Lesen und Schreiben von Speicherdaten und eine Einheit für das Auswerten von Sprungbefehlen. Die Funktionseinheiten enthalten zum Teil mehrere Pipelinestufen. Wie im Blockdiagramm angedeutet werden bei der Ausführung komplizierterer Integer- und Gleitkomma-Instruktionen die Pipelinestufen in der EX-Phase von der gleichen Instruktion mehrfach benutzt, d.h. die Instruktionen können nicht vollständig überlappt ausgeführt werden. Die EX-Phase für kompliziertere Integer-Instruktionen wie Multiplikation und Division benötigt zwischen 3 und 20 Taktzyklen. Bei den Gleitkomma-Intruktionen ist es nur die



Figur 11: Pipeline des PowerPC 620

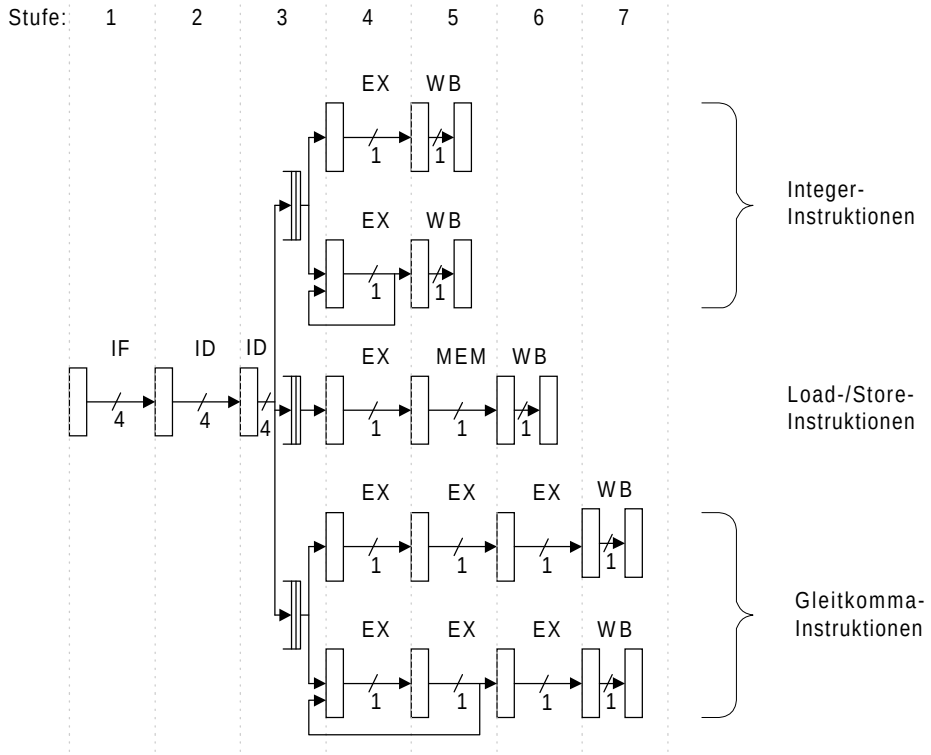
Division, die nicht überlappt ausgeführt werden kann und insgesamt 32 Taktzyklen (64-bit-Operanden) in der EX-Phase verbringt.

Figur 12 zeigt die Pipeline des 21264. Die Stufen 1 bis 4, welche für das Holen und Dekodieren der Instruktionen verantwortlich sind, können gleichzeitig vier Instruktionen verarbeiten. Von der Dekodierphase gelangen die Instruktionen in Warteschlangen, welche mit Reservationsstationen zu vergleichen sind. Es gibt zwei Warteschlangen, eine für Integer- und Load-/Store-Instruktionen und eine weitere für Gleitkomma-Instruktionen. Es stehen sechs Funktionseinheiten zur Verfügung: vier Integer-Einheiten und zwei Gleitkomma-Einheiten. Die Integer-Einheiten umfassen zwei allgemein verwendbare Einheiten und zwei Einheiten zur Berechnung von Adressen. Letztere werden beim Lesen und Schreiben von Integer- wie auch Gleitkomma-Operanden benutzt. Ebenfalls können diese beiden Einheiten einfachere arithmetische und logische Operationen ausführen. Von den beiden allgemein verwendbaren Integer-Einheiten enthält eine ein Multiplizierwerk. Weiter vorhanden sind zwei identische Gleitkomma-Einheiten. Diese führen Operationen in der Regel in vier Pipelinestufen aus. Nur aufwendigere Operationen benutzen diese Pipelinestufen in mehreren Iterationen, so benötigt beispielsweise eine 64-bit-Division 16 Taktzyklen. Die Multiplikation hingegen kann vollständig überlappt ausgeführt werden.



Figur 12: Pipeline des 21264

Die Pipeline des R10000 ist in Figur 13 gezeigt. In der Pipelinestufe 1 werden gleichzeitig vier neue Instruktionen geholt. In der Stufe 2 werden die Instruktionen dekodiert. Im Falle von bedingten und unbedingten Sprungbefehlen werden in dieser Stufe auch die Sprungadressen berechnet. In Stufe 3 werden die Instruktionen in Warteschlangen geschrieben, wo sie solange verbleiben, bis ihre Operanden verfügbar sind. Sobald die Operanden verfügbar sind, werden die entsprechenden Register zugegriffen und die Instruktion wird einer der fünf Funktionseinheiten zugeführt, wo die eigentliche Ausführung der Instruktion in Stufe 4 beginnt. Nach Abschluss der EX-Phase werden die berechneten Resultate in der ersten Hälfte der darauffolgenden Stufe in die Register geschrieben. Wie im Blockdiagramm gezeigt gibt es drei Warteschlangen, je eine für Integer-, Load-/Store- und Gleitkomma-Instruktionen. Während jeweils insgesamt vier Instruktionen pro Takt in die



Figur 13: Pipeline des R10000

Warteschlangen eingefügt werden, können pro Takt bis zu fünf Instruktionen den Warteschlangen entnommen werden.

Kompliziertere Integer- und Gleitkomma-Instruktionen werden wieder mittels eines iterativen Ansatzes berechnet. So weisen 64-bit-Integer-Multiplikation und -Division eine Latenzzeit von 10 respektive 67 Taktzyklen auf, während die 64-bit-Gleitkomma-Division eine Verzögerung von 21 Zyklen erfährt – die Ausführung einer Gleitkomma-Multiplikation kann überlappt erfolgen.

8. Ausblick

Die Leistung von Prozessoren kann mit einer Zuwachsrate von 50 % pro Jahr nur dann weiterhin gesteigert werden, falls es auch gelingt, deren Architektur weiter zu verbessern. Ansonsten, wenn lediglich die technologischen Fortschritte verbleiben, wird die Zuwachsrate auf etwa die Hälfte pro Jahr abfallen.

Weitere Innovationen werden also nötig sein, insbesondere um noch mehr Instruktionen parallel verarbeiten zu können. Dabei bildet das Erkennen von Instruktionen, welche parallel ausgeführt werden können, das Hauptproblem. Da diesbezüglich die Grenzen superskalarer Prozessorarchitekturen bereits heute erreicht zu sein scheinen, wird momentan Hoffnung gesetzt in Architekturen mit einem *Very Long Instruction Word* (VLIW). Bei diesen Architekturen wird sozusagen dem Compiler die Kontrolle über eine superskalare Pipeline gegeben: ein „langes Instruktionsformat“ ermöglicht es anzugeben, welche Instruktionen parallel ausgeführt werden sollen. Während solche Architekturen bisher v.a. bei der Signalverarbeitung zum Zuge kamen, erscheinen diese neuerdings auch attraktiv beim Bau allgemein gebräuchlicher Prozessoren. Ein Beispiel ist der Prozessor Merced, der gemeinsam von Intel und HP entwickelt wird und als möglicher Nachfolger des Pentium-Prozessors genannt wird.

Da der mittels superskalaren oder VLIW-Pipelines erreichte Parallelismus in der Regel auf ein paar wenige parallel arbeitende Funktionseinheiten beschränkt bleiben dürfte, sind längerfristig Architekturen gefragt, die sich grundsätzlicher von herkömmlichen unterscheiden. Eine mögliche Alternative könnten Architekturen sein, welche mehrere unabhängige Prozesse parallel auf einem Prozessorchip verarbeiten können [1, 19]. Natürlich verarbeitet ein solcher Prozessor den einzelnen Prozess nicht schneller, nur der gesamte Durchsatz wird erhöht.

Ein Rechnersystem soll ausgewogen sein, d.h. die einzelnen Komponenten und insbesondere auch deren Leistungsfähigkeit sollen aufeinander abgestimmt sein. Wird also die Leistung der Prozessoren gesteigert, so werden auch leistungsfähigere Speicher- und Ein-/Ausgabesysteme benötigt. Gerade die Verbesserung der Leistungsfähigkeit von Hauptspeicherbausteinen hält nicht Schritt mit den Ansprüchen moderner Prozessoren, weshalb immer ausgeklügeltere Speicherhierarchien eingesetzt werden müssen. Auch dieser Problematik ist Beachtung zu schenken und wahrscheinlich letzten Endes mit neuen Systemarchitekturen zu begegnen. Ein möglicher Weg könnte die Integration von Mikroprozessor und Hauptspeicher auf einem Chip sein, wie dies in [13] vorgeschlagen wird.

Referenzen

[1] G. Byrd, M. Holliday: *Multithreaded Processor Architectures*. IEEE Spectrum, August 1995, pp. 38-46.

[2] Digital Equipment Corporation: *Digital Technical Journal*, vol. 4, no. 4, Special Issue, 1992.

[3] F. Faggin, M. Hoff Jr., S. Mazor, M. Shima: *The History of the 4004*. IEEE Micro, vol. 16, no. 6, Dezember 1996, pp. 10-20.

[4] M. Flynn: *Computer Architecture – Pipelined and Parallel Processor Design*. Jones and Bartlett, 1995.

[5] L. Gwenap: *Digital 21264 Sets New Standard*. Microprocessor Report, vol. 10, no. 14, October 1996.

[6] A. Grove: *A Revolution in Progress*. Comdex ’96. November 1996, http:://www.intel.com/pressroom/archive/speeches/ag111.896.html.

[7] J. Heinrich: *MIPS R4000 User’s Manual*. Prentice Hall, 1993.

[8] J. Hennesy, D. Patterson: *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 1996.

[9] International Business Machines: *Journal of Research and Development*. vol. 38. no. 5, September 1994.

[10] International Solid State Circuits Conference ISSCC ’97, Februar 1997, http://www.sscs.org/isscc/.

[11] J. Keller: The 21264: *A Superscalar Alpha Processor with Out-of-Order Execution*. Microprocessor Forum, Oktober 22-23, 1996, http://www.digital.com/info/semiconductor/a2641up1/index.html.

[12] Motorola: *PowerPC 620 RISC Microprocessor Technical Summary*. http://www.mot.com/SPS/PowerPC/library/library.html.

[13] D. Patterson et al.: *A Case for Intelligent RAM*. IEEE Micro, vol. 7, no. 2, March/April 1997, pp. 34-44.

[14] R. Sites: *Alpha Architecture Reference Manual*. Digital Press, 1992.

[15] H. Stone: *High-Performance Computer Architecture*. Addison-Wesley, 1993.

[16] J. Thornton: *Design of a Computer: The Control Data 6600*. Scott, Foresman and Co., 1970.

[17] R. Tomasulo: *An Efficient Algorithm for Exploiting Multiple Arithmetic Units*. IBM Journal of Research, vol. 11, no. 1, Januar 1967.

[18] N. Tredennick: *Microprocessor-Based Computers*. IEEE Computer, vol. 29, no. 10, Oktober 1996, pp. 27-37.

[19] D. Tullsen, S. Eggers, J. Emer, H. Levy, J. Lo, R. Stamm: *Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor*. 23rd Annual International Symposium on Computer Architecture, Philadelphia, May 1996.

[20] S. Weiss, J. Smith: *POWER and PowerPC: Principles, Architecture, Implementation*. Morgan Kaufmann, 1994.

[21] World Wide Web: *CPU Info Center*. http://infopad.berkeley.eecs.edu/burd/~gpp/cpu.html.

[22] K. Yeager: *The MIPS R10000 Superscalar Microprocessor*. IEEE Micro, vol. 16, no. 2, April 1996, pp. 28-40.